计算机科学2000Vol.27№.10

CORBA 负载平衡的研究

Study on CORBA Load Balancing

骆志刚 唐雪飞 刘锦德

(电子科技大学计算机学院 成都610054)

Abstract CORBA is a widely accepted distributed computing specification, on which interoperability between heterogeneous computer platforms can be accomplished. The performance of CORBA-based application system is strongly dependent on object location and manual assignment of objects is impractial. In order to improve performance of CORBA-based distributed system, load-balancing problem must be considered. Firstly, two approaches that can be used to design load-balanced CORBA application are discussed in detail. On that basis, a general model of dynamic load balancing based on trading service is proposed. Not only load balancing is implemented in this model, but also the extension of trading service is transparent to application developers and the original applications can be integrated with it without any modification. At the end, analysis of scalability of this model is given.

Keywords CORBA, Load, Load balancing, Trading service

一、引言

作为分布式计算的一个重要规范—CORBA[1]，其主要目标是解决面向对象的异构应用之间的互操作问题，并提供了分布式计算所需的多项服务。ORB 是 CORBA 平台的核心，它用于屏蔽与底层平台有关的细节，使开发者可以集中精力去解决与应用相关的问题，而不必自己为创建分布式计算基础平台而操心。

CORBA 应用包含了大量分布在不同环境中的对象，由其组成一个复杂的分布式系统。为了提高整个系统的性能，负载平衡必然成为 CORBA 应用系统需要考虑的问题。大量的研究[2,3,4]表明，使用负载平衡策略可以提高分布式系统的性能。在 CORBA 的应用系统中，负载平衡主要是指将客户的请求在分布于不同主机和进程上的服务对象中进行分派。这一点区别于以前曾被大量研究的面向进程的分布式系统。负载平衡的粒度从进程变成了对象，这一变化导致了更频繁的通信。另外，CORBA 应用的运行环境差别很大，可以分布在不同的硬件结构和不同的操作系统上，具有很好的透明性，这一点是以前的分布式系统很难做到的。各种透明性的增加，导致系统开销的增大。除了这些差别之外，CORBA 应用系统可以借鉴以前的分布式系统负载平衡的研究成果，再加上自身的特性来实现负载平衡。

本文首先详细讨论了 CORBA 负载平衡的两种方式：应用分区法和复制法[3]。在此基础上，给出了一个基于交易服务[4]的通用动态负载平衡模型，它不仅实现了 CORBA 负载平衡，而且其交易服务的扩展对用户透明，原有应用可以直接与之集成。最后对该模型的伸缩性进行了分析，指出该模型具有良好的伸缩性，可以支持大规模的分布式系统。

二、应用分区法

应用分区法为 CORBA 应用系统所特有，它将一个应用划分为若干个相互独立的服务对象，每个服务对象提供应用的一部份功能，且互不重叠。所有分区的功能之总和为一个完整的应用。应用的划分可以按水平或垂直方向进行。水平划分主要针对应用系统的功能，每一个服务器只提供整个应用系统功能的一个子集，且相互不重叠。每个客户的服务请求只能分派给唯一一个专用分区(服务器)，其它的分区都不能完成该服务请求。这样应用系统的负载就被分布在不同的服务器上。图1给出了水平划分的示例。该应用将学生信息服务器和教师信息服务器分割开来，这样就将学生信息处理请求和教师信息处理请求严格分开。

垂直划分则主要针对应用的数据。每个服务器提供所有服务，但只能访问一部份数据。这种分法主要针对数据系统。图2给出了垂直划分的示例。两个学生信息服务器提供完整的服务，但是访问互不重叠的数据，一个只能访问一、二年级的学生信息，另一个只能访问

三、四年级的学生信息。

应用分区法直观有效，但其困难在于如何将应用系统划分为适当的分区，使得负载的分布较为公平。这就要求准确掌握应用的使用情况，如客户请求的时间倾向性和功能倾向性。同时，这是一种静态方法，对系统的动态变化不能做出恰当的反应。

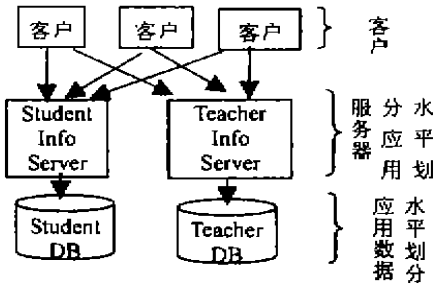

图1 水平分区

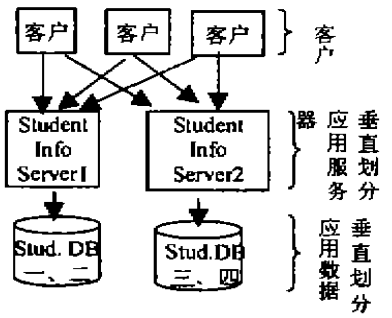

图2 垂直分区

三、复制法

复制法亦即多服务器法，使用多个服务器来完成客户的请求，每个服务器提供相同的功能，访问相同的数据。这种方法在面向进程的分布式系统中，研究很多，许多负载平衡算法可以直接应用于 CORBA 应用系统。以下我们将从对象的定位、对象的转移和对象状态的管理三个方面，对应用于 CORBA 应用系统的复制法进行讨论。

3.1 对象定位策略和机制

对象定位策略的作用是为客户的服务请求选择合适的服务器。通常，主要使用以下几种策略：

随机：服务器按随机方式进行选择，不考虑任何服务器的状态信息。这样带来的问题是所选择的服务器可能正处于忙状态，服务请求得不到及时的处理，失去负载平衡的意义。

轮询：服务器按轮询方式进行选择，也不考虑任何服务器的状态信息，其缺点如随机策略。

基于负载：这种方式考虑服务器的负载状态信息，较为常用的门槛策略和最短策略。这种方式尽管花费在负载信息收集上的代价较大，但是实践表明，这是最有效的策略[4,5]。

对象定位机制的作用是如何实现为客户的服务请求选择合适的服务器，实际应用采用了以下几种机制：

多代理(multiproxy)机制：多代理主要通过在每个客户上缓存多个有效的服务对象引用，来实现客户到不同服务实现之间的重定向机制。通常，在客户空间内，将一个可互操作对象引用 IOR(Interoperable Object Reference)构造成一个代理(proxy)，一个代理对象仅对应一个远程对象实现。多代理扩展了这一模式，允许一个代理对象在生命周期的不同时间段内，引用不同的对象实现。由于当前的 IOR 还不支持多代理，需要客户端用一个类来实现多代理，负载平衡算法则包含在多代理内，且对用户透明。客户程序使用多代理代替普通代理，由多代理完成负载平衡。多代理的优点在于简单，不需要服务器状态信息，且负载平衡的粒度可以控制到每个服务调用操作，但其效果很大程度上取决于服务的访问模式，对无状态服务器较为有效。

对象组机制：对象组是指用一个名字映射为多个对象引用，通常通过名字服务使用。它扩展了名字服务一对一的关系，允许多个对象使用同一个名字，而在命名上下文中使用了随机或轮询方式来选择服务对象。这种方式需要对名字服务进行扩展，允许多个 IOR 加入到一个对象组中，而以一个单独的名字呈现给客户。在实现时，客户应按服务方预先定义的访问模式，访问对象。这种方式透明性较好，名字服务的扩展，对客户和服务器透明，但是它没有考虑服务器的负载变化，效果有限。

选择器机制：选择器方式使用一个单独服务器来完成客户到最轻负载的服务器的定位。它从各个服务器收集负载信息，并根据这些信息，应用一定负载平衡算法，对客户的服务请求进行定位。相比前两种机制，这种方式考虑了服务器的负载信息，具有很好的动态性，其应用效果最好，但是这种方式有一个致命的弱点，选择器无规范可循，客户和服务器与选择器的交互具有特殊性，这样可能失去 CORBA 应用具有的互操作性。

3.2 迁移策略和机制

迁移策略的作用是解决系统在什么时候选择一个服务器(这里的迁移并非指一个对象从一个结点到另

个结点的移动）它控制负载平衡发生的粒度，主要包括了以下几种策略：

每个操作（per operation）：对客户每调用一次操作，都需要首先定位新的服务器，这种策略，有很好的可调粒度，但是其缺点也是明显，即频繁定位服务器的代价很大，同时保证操作间幂等性等依赖性非困难。

每个事务（per transaction）：将一个完整的 ACID 事务作为负载平衡的粒度，每个事务可能包含多个事务性的服务调用操作，在一个事务开始前，首先完成所服务器的定位，这种策略的灵活性依赖于负载平衡和一个事务的平均持续时间的相对代价。

每个会话（per session）：一旦客户被指定给一个服务器，则所有的远程调用请求都定向于该服务器，这种方式实现较为简单，且比较有效。

迁移机制的作用是如何实现迁移策略，它通常和定位机制相结合，来提供一个完整的负载平衡方案，常用的迁移机制有如下几种：

客户控制式：对象的迁移由客户控制，每个客户实现一个迁移策略来决定什么时候定位一个新的服务器。如使用每个操作策略，则在每个服务调用操作前进行服务器选择。多代理定位机制可以和每个操作策略相结合，由于频繁定位服务器仅发生在本地，这种方式的代价较小；而对象组定位机制或选择器定位机制和每个会话策略相结合，用于非频繁定位服务器，这样有效地减小服务器定位的代价。一种很好的方式是多代理定位机制和对象组定位机制相结合，同时使用每个会话策略，这种方式结合了以上几种方式的优点，既可频繁定位服务器，也使其代价降低。

服务器控制式：这种方式主要通过 CORBA 规范定义 LOCATION_FORWARD 机制，在 POA 中，允许服务器重定位客户的请求到另一个服务器。当服务器过载时，可以使用这一机制完成客户请求的重定位。这种方式不需要单个选择器收集负载信息，而由服务器相互合作来管理应用负载，但是这种方式的缺点在于使服务器复杂化，且服务器的重定位能力有限，每个服务器仅与一个对等结点有联系。

集中器控制式：集中器存在于客户和服务器之间，用于转发客户的请求到服务器，这种方式灵活，但是它的存在使 CORBA 应用完全丧失了互操作性。

3.3 状态管理

对于无状态服务器，复制法的使用简单有效。所有的服务器提供相同的功能，且服务器之间无需状态同步，但是对于有状态服务器，服务器之间需要使用同步机制进行状态同步。通常使用的同步机制有：无缓存，写副缓存和验证缓存。无论那一种同步机制，代价都很大，所以在选择迁移策略时，应考虑是否会造成大量状态同步。

四、一个基于交易服务的动态负载平衡模型

基于以上的讨论，结合 CORBA 交易服务的灵活性，易扩展性，本文提出了一个通用的负载平衡模型。交易服务是按类型和属性查找服务，比名字服务更加灵活。同一类型的服务可以对应多个具有相同功能的对象，不必象名字服务要引入对象组，才能解决一对多的问题，也不必为实现负载平衡单独增加一个选择器，并且不会增加客户和服务方的复杂性。交易器的扩展，对客户和服务器透明。

4.1 交易器的扩展

CORBA 应用都作为对象而存在，其中任何一个对象都可以向其它对象提供服务，同时又可以请求其它对象所能提供的服务。服务方以界面的形式向客户方说明自己所能提供的服务（由操作集表示），交易器接收并登记服务方的信息，向系统中潜在的客户方宣布相应服务器所能提供的服务；并能够应答客户请求，向客户提供它们所需的服务方的连接信息。

为了通过交易器实现负载平衡，交易器中需要存放相关的负载信息。由于负载信息需要频繁更新，因此我们将它从服务属性中独立出来。在交易器中增加单独的负载信息模块完成对负载信息的收集、存放和管理等操作。扩展后的交易器结构如图3所示，图3中各个组成部分为：

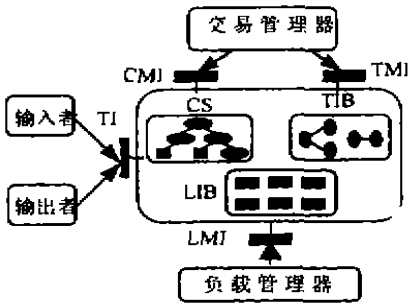

图3 交易器的扩展

·上下文空间（CS）：存放交易器收到的所有服务提交，其中的每一个上下文都包含一个服务提交队列，用以存放在该上下文中登记的服务提交。

·类型信息库（TIB）：存放在交易器登记的所有服务类型，每个服务对象都是其中某服务类型的实例。

·负载信息库（LIB）：存放负载相关的信息，其中

包含主机名、负载指标和主机上的服务器等。

扩展后的交易器定义了以下界面：

·交易界面(TI)：定义应用程序可以直接使用的操作。输入者/输出者利用这一界面输入(import)/输出(export)服务提交。

·上下文管理界面(CMI)：用于管理交易器内部的组织，提供上下文管理操作。

·类型管理界面(TMI)：用于管理交易器的类型信息，例如：登记或注销某服务类型。

·负载管理界面(LMI)：设置负载平衡策略，调整负载平衡算法的各种参数，完成对负载信息的收集。

对于负载信息策略，我们采用了周期策略和状态改变策略相结合的方式。亦即，交易器定期(周期P)向各个服务器所在主机收集负载信息；同时各个服务器所在主机的负载服务器在状态改变时，主动向交易器上载自己的负载信息。这样，交易器收集负载信息时，如果发现在周期P内负载信息已作更新，则不再向相应主机查询，由此，交易器的负担得到相应的减小。

4.2 负载平衡系统的组成及描述

图4给出了基于交易服务的负载平衡系统的组成。与普通系统比较而言，除了交易器本身的扩展外，在服务器所在主机上需要增加一个单独的程序——负载服务器LoadSrv，来完成对本地负载信息的收集。LoadSrv可以作为应用的一部分，以一个单独线程的形式出现，但这样有两个不利因素：①增加应用的复杂性，原有的应用需要改造；②一台主机上有多个服务器，造成LoadSrv的重复。因此，我们采用在服务器主机上单独运行LoadSrv的方式。

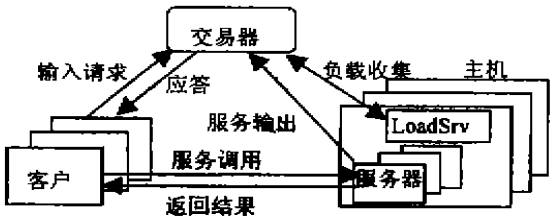

图4 负载平衡系统

为了对整个系统的各个角色描述，定义以下几个集合：集合$T=\{T_1,T_2,\cdots,T_n\}$表示系统中有n个可用的服务类型；集合$S=\{S_1,S_2,\cdots,S_m\}$表示系统中有m个服务器；集合$C=\{C_1,C_2,\cdots,C_q\}$表示系统中有q个客户；集合$N=\{N_1,N_2,\cdots,N_w\}$表示服务器所在主机有w台；而$L\subset R^+$为负载指标的值域。

对于服务器，其行为可描述为：

$p:S\rightarrow F(T)$

它表示服务器到服务类型的映射，其中F(T)表示T的超集，即集合T的所有子集组成的集合，例如$p(S_i)=\{T_{i,1},T_{i,2},\cdots,T_{i,t}\}$，表示$S_i$可以提供的服务类型为$\{T_{i,1},T_{i,2},\cdots,T_{i,t}\}$。服务器向交易器输出服务时，使用该映射完成服务到类型的映射，从而登记在交易器中。

对于交易器，其行为可描述为：

$t:T\rightarrow F(S)$

它表示服务类型到服务器的映射，例如：$t(T_1)=\{S_{n,1},S_{n,2},\cdots,S_{n,u}\}$表示当前有这些服务器：$\{S_{n,1},S_{n,2},\cdots,S_{n,u}\}$支持服务类型$T_1$，t实际包含了输入、输出和撤消等操作。

在交易器中增加了负载信息模块后，增加了以下一对行为：

$h:S\rightarrow N$ 和 $l:N\rightarrow L$

它们分别表示服务器到所在主机的映射和主机到其负载的映射。交易器在执行负载平衡操作时，使用这些映射。

对于客户，其行为描述为：

$r:C\rightarrow F(T)$

它表示客户所需服务类型，例如：$r(C_i)=\{T_{C_i,1},T_{C_i,2},\cdots,T_{C_i,w}\}$表示客户$C_i$需要这些服务类型$\{T_{C_i,1},T_{C_i,2},\cdots,T_{C_i,w}\}$的服务。客户在向交易器请求输入服务时，需要这一映射，向交易表达所需的服务类型。

对于负载服务器(LoadSrv)，行为描述如下：

$c:N\rightarrow L$

它表示对所在主机进行负载信息收集。

从以上的描述也可以看出，交易器进行扩展后，对客户和服务方都是透明的，不会影响原有的操作，扩展部分集中在交易器和新增加的负载服务器上。

4.3 模型伸缩性

从性能和交易器扩展后的透明性两方面考虑，集中式算法[2,6]比较适合交易器的模型，但其伸缩性如何将直接影响其实用性。要使其具有较好的伸缩性，需要在系统到达一定规模时，交易器的最大操作时间要控制在应用可以接受的范围。

交易器中的操作主要包括：输出、输入及负载信息的收集。这些操作的时间有所不同。设d为其中的最大操作时间(实际上，负载信息收集的时间最大)，整个负载平衡系统有N台主机(这里包括了客户主机和服务器主机)，每台主机的请求率为λ，则交易器的请求到达率为Nλ，我们用M/D/1作为交易器请求的排队模型[6]，则交易器的请求的平均延迟(排队论中称为平均逗留时间)为：

$$\bar{D}=d+N\lambda d^2/2(1-N\lambda d) \qquad (1)$$

给定一可接受的$\bar{D}_a(\bar{D}\leqslant\bar{D}_a)$，则可以通过(1)式求出交易器可以支持的最多结点数：

$$N_{max}=\left[\frac{2(\overline{D_a}-d)}{d\lambda(2\overline{D_a}-d)}\right] \quad (2)$$

实际上，交易器的所有操作中，负载信息收集的时间最长（约3ms，其中约2ms 的时间用在通信上，对交易器所在主机没有特别的要求，一般的主机即可），令 $d=0.003s$，$\overline{D_a}=0.1s$，$\lambda=0.9$，通过(2)式可以计算出 $N_{max}=361$，即一个交易器可以支持364个结点。对于更大规模的分布式系统，可以采用以下几种方法扩大交易器的伸缩性：①交易器使用多线程的方式，特别是负载信息的收集可以采用单独的线程完成；②可以使用联合交易的方式（这是交易器的灵活表现之一），将系统划分为不同的域，由不同的交易器处理不同域内的请求，同时交易器之间通过联合交易的方式，处理跨域之间的请求；③由于 CORBA 的 IIOP 协议使用的是 TCP 协议，通信成为交易器的伸缩性的瓶颈，采用更快的网络也是扩大规模的方法之一。

从上面的分析可以看出，集中式的方式也可以支持大规模的分布式系统，[6]中也得到了类似的结论。

结束语 随着 CORBA 应用范围日益扩大，负载平衡成了其应当考虑的问题。本文首先对 CORBA 负载平衡所使用的各种策略、机制进行了详细的讨论。在此基础上，提出了一个基于交易服务的动态负载平衡模型。本模型对于交易器的扩展，对用户透明，从透明性和性能两方面考虑，我们建议使用集中式算法。文章的最后对该模型的伸缩性进行了分析，分析指出它可以支持大规模的分布式系统。由于采用集中式的方式，交易器的可靠性对整个分布式系统至关重要，如何提高交易器的可靠性是我们下一步的工作。

参 考 文 献

1 Object Management Group Common Object Request Broker: Architecture and Specification, 2.2 ed., February 1998

2 Zhou S. A Trace-Driven Simulation Study of Dynamic Load Balancing. IEEE Trans. on Software Engineering, 1988, 14(9): 1327～1341

3 Slama D, et al. Enterprise CORBA. Prentice Hall, 1999

4 OMG. CORBAservices: Common Object Services Specification. February 1998

5 Shivararri G, et al. Load Distributing for Locally Distributed Systems. IEEE Computer, 1992, 25(Dec.): 33～44

6 Lin Hwa-Chu, Raghavendra C S. A Dynamic Load Balancing Policy with a Central Job Dispatch (LBC). IEEE Trans. on Software Engineering, 1992, 18(2): 148～157

(上接第30页)

之处会有用武之地。ODP 技术是由国际标准化组织精心策划下所开发的解决方案，它面向大型和特大型的分布式应用系统。随着时间的推移，它将受到愈来愈多的人的重视和采用。有人预见，它的 ODP 参考模型将成为 OSI 参考模型的有效补充。CORBA 技术是一个正在蓬勃发展中的解决方案，虽然 CORBA 中间件需要不断地改进和充实，但由于存在极有实效的技术社会的支撑，因而它的发展前景十分宽广。

参 考 文 献

1 Gill P J. What Is Open System? Uniforum Monthly, 1992, 12(1)

2 Hach D. Forcing an Open Systems Definition. Open Computing, 1994, 11(9)

3 刘锦德，唐雪飞. 军用计算机开放系统轮廓描述的研究：[电子科技大学研究报告]. 1992

4 Liu Jinde. Open System Connotation. 19th Intl. Computer Conf, Hong Kong, 1996

5 刘锦德. 对于开放系统内函的澄清，计算机应用. 1997, 17(6)

6 Millikin M D. DCE--Interoperability Now! Network Monitor, 1990, 5(1)

7 Gray P. Open Systems and IBM. McGraw-Hill, 1993

8 唐雪飞. 开放系统中互操作性的研究：[电子科技大学博士论文]. 1996

9 苏森，唐雪飞. 开放系统中的互操作性. 计算机应用, 1997, 17(6)

10 Gill P J. DCE Status Report. Uniforum Monthly, 1995, 15(3)

11 De Nitto K. Richer Feature Set for DCE. Uniforum Monthly, 1997, 17(1)

12 Farooqui K, et al. The ISO Reference Model for Open Distributed Processing and Standards. Computer Network and ISDN Systems. 1995, 27: 1215～1229

13 ISO/IEC & ITU. Reference Model of ODP, Part 1 Overviesw. May1995

14 刘锦德，苏森. CORBA 技术的新发展. 计算机应用, 1999, 19(5)

15 OMG. The Common Object Request Broker: Architecture and Specification, Revision 2.2. Feb. 1998

16 Orfali R, et al. Instant CORBA. John Wiley & Sons, 1997